Hierarchical Schedulers, Performance Guarantees, and Resource Management

John Regehr and Jack Stankovic
Department of Computer Science
Thornton Hall, University of Virginia
Charlottesville, VA 22903
{regehr,stankovic}@cs.virginia.edu

Introduction

An attractive approach to scheduling applications with diverse CPU scheduling requirements is to use different schedulers for different applications. For example: real-time schedulers allow applications to perform computations before deadlines, time-sharing schedulers provide high throughput for compute-bound processes and fast response time for interactive applications, and gang schedulers and cluster coschedulers permit tightly-coupled parallel applications to achieve high performance in the presence of multiprogramming. Furthermore, individual members of these broad classes of algorithms make tradeoffs that may or may not be appropriate for a given situation. In order to take advantage of these diverse algorithms, we permit schedulers to be arranged in a hierarchy - a *root scheduler* gives CPU time to the schedulers below it in the hierarchy and so on until an application thread is scheduled by a *leaf scheduler*. This architecture has a number of advantages:

- The scheduling hierarchy can be tailored to the set of applications that is running at any given time.
- Applications or groups of applications can be flexibly isolated from each other. For example, a proportional-share scheduler near top the of the hierarchy can limit applications, users, or accounting domains to some fraction of the CPU regardless of the number of threads they are running.
- Schedulers can be designed modularly and narrowly focused in order to do a good job scheduling a particular type of application.

That was the "motherhood and apple pie" introduction to hierarchical scheduling. The following two sections discuss our improvements to previous hierarchical scheduling work such as *CPU Inheritance Scheduling* by Ford and Susarla (OSDI '96) and *A Hierarchical CPU Scheduler for Multimedia Operating Systems* by Goyal, Guo, and Vin (also OSDI '96).

Performance Guarantees

Clearly, there are hierarchical arrangements of schedulers that don't make sense because they prevent schedulers from providing the scheduling properties that they are designed to provide. For example, arranging for a real-time scheduler to be scheduled by a time-sharing scheduler - since the time-sharing scheduler guarantees no minimum rate of progress to the threads (or schedulers) that it schedules, the real-time scheduler would have no basis for assuming that any thread it schedules would be able to make its deadlines. We employ *guarantees*, (potentially revocable) contracts for a particular kind of scheduling behavior, to help compose schedulers. For example, if a rate-monotonic scheduler is guaranteed (by another scheduler) to receive 5ms of CPU time every 20ms, then it will be able to schedule some kinds of tasks if their periods are also longer than 20ms. Obviously, a scheduler can provide a guarantee no stronger than the one it receives. We say that an arrangement of schedulers is *hierarchically correct* if all schedulers in the hierarchy can provide their guarantees—maintaining this property is a key issue in hierarchical scheduling.

UU IR Author Manuscript

A key issue in resource management is the tension between sharing and isolation. An application is isolated from other apps when, for example, it is given a guarantee of a minimum execution rate and granularity. Once cycles are guaranteed to some application, the schedulers are no longer free to assign them, on the fly, to some other application. In this sense, making a guarantee puts a restriction on schedulers' future decision making. Guarantees can be weakened when applications are known to gracefully degrade, or when they support renegotiation of guarantees in order to admit more important tasks.

Resource Management

Our second contribution to hierarchical scheduling is a *resource manager*, a generalized admission control system that provides a level of indirection between applications and the scheduling hierarchy. The resource manager is responsible for maintaining hierarchical correctness, for loading new schedulers into the hierarchy when needed, for mapping threads requesting a special kind of scheduling to an appropriate scheduler, and for enforcing user-specified policies about the allocation of CPU time. We enforce policies in the resource manger in order to keep them out of the schedulers themselves – to separate mechanism from policy. Policies will take the form of a rule attached to a scheduler, a user, or an application; the resource manger checks these rules before attaching a thread to a particular scheduler. For example, rules could:

- Prevent any user except the one who "owns" a particular scheduler from attaching threads or schedulers to it.
- Specify that an application can adapt to various conditions; e.g. an MPEG player might support several resolutions that each require a different amount of processor time.
- Specify application importance. If a home computer is being used to implement a voice-mail system, the user could indicate that voice mail is more important than playing games. Then, an incoming call will cause the resource manager to revoke the guarantee given to the user's Quake process in order to answer the call.

Status

We are currently beginning to implement hierarchical scheduling with performance guarantees and resource management in a release candidate of Windows 2000.

Hierarchical Schedulers, Performance Guarantees, and Resource Management

John Regehr and Jack Stankovic
University of Virginia

The Problem

- People do all sorts of things using general-purpose OSs
  - Convenience, compatibility, commodity
- CPU schedulers aren't general enough to support the apps well
  - Time-sharing
  - Hard/soft real-time
  - Server
  - Parallel

A Solution: Hierarchical Scheduling

- Applications with different scheduling requirements get different schedulers
- Top-level scheduler arbitrates among lower-level schedulers
- Enables:
  - Flexible composition of modular schedulers
  - Hierarchical load isolation (easy)
  - Flexible sharing (not as easy)

Performance Guarantees

- Schedulers require / provide guarantees
  - Contract for type of scheduling provided; e.g.
    - 5ms every 33ms
    - 200ms before deadline
    - 70% of CPU
    - proportional share of available CPU
- Used to reason about hierarchical schedulers

Composability

- What does it mean when schedulers stack?
  - Are they all going to work?
- Lots of combinations won’t work
  - Non-RT / RT
- Within RT, some combinations work
  - Depends on applications as well as schedulers

The Resource Manager

- Enforces user-specified policies attached to schedulers, users, applications; e.g.
  - Only user X can add threads to scheduler Y
  - DVD player is adaptable
  - Answering phone is more important than playing Quake
- Also:
  - Maps threads to schedulers
  - Loads new schedulers
  - Ensures composability

Status

- Beginning to implement in Windows 2000
- Let’s talk….